08/15/00 Jc880 U.S. PTO

08-16-00

A

Jc869 U.S. PTO 09/638491 08/15/00

Our Docket No.: **N298.12-0001**	Date: **August 15, 2000**
First Named Inventor: **Michael Feldman**	
Express Mail No.: **EL177000089US**	

EL177000089US

UTILITY PATENT APPLICATION TRANSMITTAL

APPLICATION ELEMENTS

ADDRESS TO: **Assistant Commissioner for Patents**
Box Patent Application
Washington, D.C. 20231

1. [X] Fee Calculation Sheet
(Submit an original and a duplicate for fee processing)
2. [X] Specification Total Pages [21]
   - Descriptive title of the invention
   - Cross References to Related Applications
   - Statement Regarding Fed. Sponsored R&D
   - Reference to Microfiche Appendix
   - Background of the Invention
   - Brief Summary of the Invention
   - Brief Description of the Drawings (if filed)
   - Detailed Description
   - Claims
   - Abstract of the Disclosure
3. [X] Drawings (35 U.S.C. 113) Total Sheets [5]
4. [X] Oath or Declaration Total Pages [2]
   a. [] Newly Executed (original or copy)
   b. [] Copy from a prior application (37 C.F.R. 1.63(d) - for continuation/divisional with Box 18 completed)
   [Mark Box 5 below]
   I. [] DELETION OF INVENTOR(S)
   Signed statement attached deleting inventor(s) named in the prior application, see 37 C.F.R. 1.63(d)(2) and 1.33(b)
5. [] Incorporation by Reference (useable if Box 4b is checked). The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 4b, is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein

6. [] Microfiche Computer Program (Appendix)
7. Nucleotide and/or Amino Acid Sequence Submission (*If applicable, all necessary*)
   a. [] Computer Readable Copy
   b. [] Paper Copy (identical to Computer Copy)
   c. [] Statement verifying identify of above copies

ACCOMPANYING APPLICATION PARTS

8. [] Assignment Papers (cover sheet & document(s))
9. [] 37 C.F.R. 3.73(b) Submission
   [] Power of Attorney
10. [] English Translation Document (*if applicable*)
11. [] Information Disclosure Statement with Copies of Citations as necessary
12. [] Preliminary Amendment Total Pages []
13. [X] Return Receipt Postcard (*Should be specifically itemized*)
14. [X] Small Entity Statement(s)
   [] Statement filed in Prior Application. Status still proper and desired
15. [] Certified Copy of Priority document(s) (*If foreign priority is claimed*)
16. [] File Data Sheet
17. [] Other

18. **If a CONTINUING APPLICATION**, *check appropriate box and supply the requisite information:*
[] Continuation [] Division [] Continuation-in-part (CIP) of prior Application No.

19. **CORRESPONDENCE ADDRESS**

ATTY NAME AND REG. NO.	**Jeff A. Holmen** **38,492** SIGNATURE: Jeff A. Holmen
ADDRESS	**Kinney & Lange, P.A.** **THE KINNEY & LANGE BUILDING** **312 South Third Street** **Minneapolis, MN 55415-1002**
TELEPHONE	**(612) 339-1863**
FAX	**(612) 339-6580**

FEE TRANSMITTAL

Complete if Known	
Application No.	
Filing Date	August 15, 2000
First Named Inventor	Michael Feldman
Group Art Unit	
Examiner Name	
Atty. Docket Number	N298.12-0001

Total Amount of Payment: $ 408.00

METHOD OF PAYMENT (Check One)

1. [X] The Commissioner is hereby authorized to charge any additional fee required under 37 C.F.R. 1.16 and 1.17 and credit any over payments to Deposit Account No.11-0982.
Deposit Account Name: Kinney & Lange, P.A.

2. [X] Check Enclosed

FEE CALCULATION

1. BASIC FILING FEE

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description
101	690	201	345	[X] Utility Filing Fee
106	310	206	155	[ ] Design Filing Fee
108	690	208	345	[ ] Reissue Filing Fee
114	150	214	75	[ ] Prov. Filing Fee

Subtotal (1) $345.00

2. EXTRA CLAIM FEES

	Number Claims		Prior**		Extra		Fee from Below		Fee Paid
Total	27	-	20	=	7	x	9	=	63.00
Indep.	3	-	3	=	0	x	__	=	0.00
Multiple Dependent Claims							*	=	*

**Insert 3 and 20, or number previously paid if greater; Reissue see below

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Description
103	18	203	9	Claims in excess of 20
102	78	202	39	Independent claims in excess of 3
104	260	204	130	Multiple Dependent Claim
109	78	209	39	Reissue Independent Claims Over Original Patent
110	18	210	9	Reissue claims in excess of 20 and over original patent

Subtotal (2) $63.00

FEE CALCULATION (Continued)

3. ADDITIONAL FEES

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee paid
105	130	205	65	Surcharge - Late filing fee or oath	*
127	50	227	25	Surcharge - late provisional filing fee or cover sheet	*
139	130	139	130	Non-English specification	*
147	2,520	147	2,520	For Filing a Request for Reexamination	*
115	110	215	55	Extension for reply within first month	*
116	380	216	190	Extension for reply within second month	*
117	870	217	435	Extension for reply within third month	*
118	1,360	218	690	Extension for reply within fourth month	*
128	1,850	280	925	Extension for reply within fifth month	*
120	300	220	150	Filing a brief in support of an appeal	*
121	260	221	130	Request for oral hearing	*
148	110	248	55	Terminal Disclaimer Fee	*
140	110	240	55	Petition to revive - unavoidable	*
141	1,320	241	660	Petition to revive - unintentional	*
142	1,240	242	635	Utility/Reissue issue fee (inc. advance copies)	*
143	460	243	245	Design issue fee (inc. advance copies)	*
122	130	122	130	Petitions to the Commissioner	*
123	50	123	50	Petitions related to provisional applications	*
126	240	126	240	Submission of Information Disclosure Statement	*
581	40	581	40	Recording each patent assignment per property (times number of properties)	*
Other fee (specify) ____					*

Subtotal (3) $0.00

Signature Jeff A. Holmen

Jeff A. Holmen

Reg. No. 38,492

Date ____

Deposit Account No. 11-0982

STATEMENT OF SMALL ENTITY STATUS (SMALL BUSINESS CONCERN)	Attorney Docket No. N298.12-0001

First Named Inventor : Michael Feldman

Title : SYSTEM AND METHOD FOR GENERATING DISTRIBUTED INFORMATION SYSTEMS

With respect to the invention described in:

X the application filed herewith.

__ Application No. ________, filed ________.

__ Patent No. ________, issued ________.

I. STATEMENT OF QUALIFICATION AS A SMALL ENTITY

I am:

__ the owner of the small business concern identified below:

X an official of the small business concern empowered to act on behalf of the concern identified below:

NAME OF CONCERN : Nash Controlware, Inc.
ADDRESS OF CONCERN : 7386 Washington Avenue South
Eden Prairie, MN 55344

The above-identified small business concern qualifies as a small business concern as defined in 13 C.F.R. 121.12, and reproduced in 37 C.F.R. 1.9(d), for purposes of paying reduced fees under 35 U.S.C. 41(a) and (b).

II. STATEMENT OF OWNERSHIP

Rights under contract or law remain with or have been conveyed to the above-identified concern. If the rights held are not exclusive, each individual, concern or organization having rights to the invention is listed below and no rights to the invention are held by any person who could not be classified as (1) an independent inventor under 37 C.F.R. 1.9(c) if that person had made the invention, (2) a small business concern under 37 C.F.R. 1.9(d) or (3) a non-profit organization under 37 C.F.R. 1.9(e).

X There is no such person, concern or organization.

__ The person(s), concern(s) or organization(s) is listed below:

FULL NAME ____

ADDRESS ____________________

[ ] Individual
[ ] Small Business Concern
[ ] Non-Profit Organization

III. ACKNOWLEDGEMENT OF DUTY TO NOTIFY PTO OF STATUS CHANGE

I acknowledge the duty to file, in this application or patent, notification of any change resulting in loss of entitlement to small entity status pursuant to 37 C.F.R. 1.28(b).

IV. SIGNATURE

Signature: ____________________ Date: 8/15/2000
Joel Nash

Title: CEO

09518051.081500

SYSTEM AND METHOD FOR GENERATING DISTRIBUTED INFORMATION SYSTEMS

CROSS-REFERENCE TO RELATED APPLICATION(S)

This application claims the benefit of the filing date of U.S. provisional application serial number 60/149,507, entitled "SYSTEM AND METHOD FOR GENERATING DISTRIBUTED INTELLIGENCE SYSTEMS" which was filed August 17, 1999.

BACKGROUND OF THE INVENTION

Developers of distributed information systems are faced with daunting complexities. The traditional approach to system design and development requires a monumental task of understanding constantly changing requirements, while designing and implementing the system as a whole. The requirements for the system are collected and interpreted by software developers who are not the domain experts. At the same time, people who have an intimate knowledge of the system requirements are not the software engineers.

There are several inherent problems with existing approaches:

1. Multi-phased design and development process that is not extensible.

Difficulty in communication between developers and domain experts results in multiple iterations of a system design and multiple patches and changes to the delivered product. Such a system, when completed, becomes a legacy island in the enterprise that is impossible to change, extend, or integrate into the global information infrastructure.

Prior art solutions have tried to solve this major problem by introducing new languages targeted to capture requirements for the system design, such as the graphical "Use Cases" language of UML. These new languages add an extra level of complexity and require a high level of commitment from both groups involved in the design and development process. The biggest problem with this approach is that the design model is not present in the system delivered to the

customer. An end user gets a system that consists of files, modules, and executables, but not of accounts, machines, units, etc. From the end users' standpoint, all of the time that went into the requirements capturing and modeling was wasted, because the system does not represent their real-world entities that they interact with, but some foreign entities forced on them by the system implementation.

This prior art approach does not help developers simplify the design and implementation of the system. Developers have to deal with the details of a target deployment environment, communication and hardware. An object-oriented approach to the system implementation, while helping in the design process, leaves them with monolithic applications once compiled.

2. Changes are difficult to make.

This application orientation makes prior art approaches much more difficult to use in the environments where requirements are constantly changing and system complexities are increasing. Even component specifications that have been introduced did not address the distributed nature of the systems, nor did they help to solve the complexities of the development process, and were a mere extension of the client-server model of the past.

3. Communication between applications is limited.

Monolithic applications have no way to interact with other applications deployed in the enterprise. A special integration infrastructure has to be used to build an integration layer to pull it all together. This integration is an afterthought solution that is an application by itself and has all the problems noted above.

4. Difficulty in transferring large amounts of data to a central point.

With traditionally developed information systems, decision-making is centralized even though the information sources are distributed throughout the enterprise. Generally, information is transferred to a central point where it is processed. In physically distributed enterprises, with either large buildings or worldwide operations, it is very difficult to transfer large amounts of information to a central point. Often the solution is to install multiple copies of an application, each in an area of the enterprise. This results in unconnected islands, with little or no synchronization between areas.

5. Not designed for real-time.

Most prior art applications were not designed for real-time behavior. With the exception of real-time control systems, most applications were designed to run periodically, perhaps a few times a day or once a week to update inventory, send orders to the suppliers, or process production data for the last day or week. This limitation prevents businesses from immediately reacting to needs of customers or reacting to problems with internal operations. There is a need to have all applications, including supply chain management, e-commerce and plant-floor operations, to react in real-time as an integrated enterprise.

BRIEF SUMMARY OF THE INVENTION

An architecture for developing a distributed information system comprises a service definition tool for generating service protocols as a service definition. Each service protocol includes a plurality of messages. The messages include incoming messages and outgoing messages. Each message carries a plurality of data fields. A component development tool generates a first and a second plurality of components that implement and consume services. Each component in the first plurality of components represents a physical entity in the

distributed information system. Each component in the second plurality of components represents a logical entity in the distributed information system. A system development tool generates a plurality of component instances based on the first and the second plurality of components. An engine software program runs on each of a plurality of networked nodes. The engine software program provides a programmable run-time environment for hosting the plurality of component instances and supporting communication between component instances.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 shows a diagram of a component which provides implementation of service protocols.

FIG. 2 shows a diagram illustrating the interaction of prior art components.

FIG. 3 shows a diagram illustrating self-sufficient components according to the present invention.

FIG. 4 shows an example of a system model built from components and connected by links according to the present invention.

FIG. 5 shows a system infrastructure according to the present invention.

FIG. 6 shows an architecture for creating and managing distributed information systems according to the present invention.

FIG. 7 shows a diagram illustrating the deployment of component instances and links to nodes.

FIG. 8 shows a diagram illustrating the interaction between a component instance and run-time software.

DETAILED DESCRIPTION

The architecture presented by this invention uses components as building blocks for distributed information systems. By placing components as the centerpiece of the design and development process, this invention improves on the prior use of components as application parts that are glued together by application code. This invention makes another step toward generalization of components by defining them as service providers and consumers. A service represents an "operation" or activity that is continuous and internal to the component. Because implementation of the service is internal to the component, external entities do not have direct access to the service. External entities can interact with the service by sending messages to and receiving messages from the service implementation (component). Services are defined by protocols - collections of incoming and outgoing messages. Another way to describe service protocols is to treat incoming messages as function calls, implemented by the service, and outgoing messages as events raised by the service. Service providers are responsible for implementing handlers for incoming messages as defined by the service protocol. Service consumers are responsible for implementing handlers for outgoing messages as defined by the protocol. Any consumer can use any provider if they implement the same protocol. This allows components to be modeled as collections of provided and consumed services. For example, a product routing component can implement routing service functionality and consume equipment service functionality of components representing factory floor equipment. Components can provide or consume any number of services. This adds flexibility to components and allows a system approach to development.

FIG. 1 shows component 10, which provides implementation of service protocols by exposing ports 11 and 12. Component 10 may implement any number of service protocols as service provider and/or as service consumer. When a component 10 implements a service protocol, it exposes an access port. There are

service provider port 11 and service consumer port 12 implementations of access ports, depending on required functionality. Service protocols are always defined from the provider point of view. Consumer implementation reverses direction of messages as defined in the protocol, e.g. incoming messages become outgoing, and outgoing messages are coming in. System development tools use access ports to identify available end points for the possible interconnections between component instances.

The following is an example of a service protocol definition in XML (Extensible Markup Language):

```
<Service Name = 'Mixing Station'>
        <In>
                <Message Name = 'Start'>
                        <Parameter Name = 'Duration', Type = long />
                </Message>
                <Message Name = 'Stop' />
        </In>
        <Out>
                <Message Name = 'Status'>
                        <Parameter Name = 'Elapsed_Time', Type = long />
                        <Parameter Name = 'Level', Type = double />
                        <Parameter Name = 'Error_Code', Type = Errors/>
                </Message>
        </Out>
        <Type Name = 'Errors', Type = enum>
                <Field Name = 'None', Value = 0 />
                <Field Name = 'Empty', Value = 1 />
                <Field Name = 'Motor_Failed', Value = 2 />
                <Field Name = 'Cycle_Completed', Value = 3 />
```

</Type>

</Service>

This example depicts a protocol for the service "Mixing Station" that has two incoming messages, "Start" and "Stop", where the "Start" message carries the parameter "Duration" of the type "long". It also has one outgoing message "Status" with three parameters - "Elapsed_Time" of the type "long", "Level" of the type "double", and "Error_Code" of the locally defined type "Errors" that can be one of the following values: "None", "Empty", "Motor_Failed", or "Cycle_Completed".

Service protocols are different from interfaces as defined by DCE (Distributed Computing Environment) RPC (Remote Procedure Call), COM (Component Object Model) /DCOM (Distributed COM), CORBA (Common Object Request Broker Architecture) and Java RMI (Remote Method Invocation). Service protocols, according to the present invention, assume an asynchronous, bidirectional communication model, unlike the synchronous, unidirectional, RPC-based model of the above-mentioned specifications. This invention's approach frees components from being dependent on the knowledge of a peer component, but more importantly, components are not dependent on the presence of a peer at all. These prior art specifications are based on an assumption of one component being a client of the other component. FIG. 2 represents the interaction of prior art components 21 and 22, where activity 23 can occur only when the two components 21 and 22 interact. Component 22 has no knowledge of the capabilities of component 21; component 22 is a server and component 21 is a client. Communication between components 21 and 22 is unidirectional as represented by arrow 24. Communication is initiated with an RPC call. Activity 23 exists only in the context of the RPC call from component 21 to component 22. In other words, component 21 has to get a reference to the peer component 22 or to the proxy of the peer by creating it, or by some other means. This prior art approach also implies that one component cannot work without other components present online. That is,

any activity 23 within a system can occur only when components interact. In a distributed, multi-node system, this requirement is impossible to satisfy without going into extreme hardware and network solutions that are expensive, proprietary and cannot be cost-effectively deployed on a large scale. This also limits what can be modeled using this approach. Most real-world objects operate on a continuous basis, concurrently, not just during function calls, which forces developers to emulate concurrence in their components when developing for existing specifications.

FIG. 3 shows how the invention's components can interact while being self-sufficient. Components 31 and 32 have corresponding activities 35 and 34. Exchanging messages over link 33 creates additional activities 36. This invention's components 31 and 32 are different from the prior art in that they are designed to operate as stand-alone entities, with no knowledge of peers or proxies, executing their own activities 34 and 35. A run-time environment handles all communication details for the components 31 and 32. Service consumers can come and go, as they will, without affecting functionality of the service providers. This also shifts design decisions by the component developer from functionality of the system as a whole to functionality of the component. For example, a component monitoring production output can be designed and built without knowledge of the system it will be used in. Communication between components is defined not by the component developer, but by the system developer and can be changed at any time without affecting the components themselves. Communication is accomplished by creating a link 33 between a service provider component port 11 and a complimentary service consumer component port 12 (see FIG. 1). Link implementation, provided by the system run-time, is responsible for delivering messages between connected ports. A link 33 can be created between two ports of the same component. A port can have an unlimited number of links 33 connected to it, such supporting one to many, many to one and many to many patterns.

Interacting components create additional, high level activities that implement desired system functionality 36.

An effect of this inventive approach is simplification of system design. Because each component is a stand-alone entity, it can be designed, implemented and tested stand-alone. This greatly simplifies testing and debugging of the system because there is no additional 'glue' code to test and debug. It also promotes a common, domain specific terminology use within a system. For example, a control solution may use components such as sensors, pumps and valves, where a MES (Manufacturing Execution System) solution may use BOM (Bill Of Materials), inventory and work cell components. Collaboration between developers and domain experts is simplified because of this and there no need for yet another language to use.

In the real world, entities modeled by components are parts of a hierarchical structure, where components on the different levels are dependent on other components in the hierarchy. The old approach for modeling this - decomposition, where the whole system is modeled and then components are built as parts of the whole, produces non- portable and inflexible solutions. This is a top to bottom approach. This invention reverses this approach by modeling from bottom up. This makes a lot of sense because bottom level components are more generic than components on the higher levels of a hierarchy. For example, in an industrial control system, components such as sensors, valves, motors, etc are generic, where components directly related to the process implemented are specific to that process. In a MES system, generic components are: inventory item, work cell, final product, etc.; and non-generic components are: process manager, production sequencer, and BOM. FIG. 4 shows an example of a system model 40 built from components 41A-41I (collectively referred to as components 41) connected by links 42A-42I (collectively referred to as links 42). By building libraries of generic components 41, new systems can be created with minimal new

development efforts and improved reliability by defining components 41 and linking them together with links 42.

Users, building solutions as defined by this invention, do not deal with applications any more - they work with the system as a whole. This is again in contrast to the prior art solutions where distributed systems are built of multiple applications. Tools, targeting domain experts/users, reinforce and promote this approach to system development. Because there is not a monolithic application anywhere in the system, but a hierarchy of components, system tools can represent a user with the picture of the system as it was originally modeled. This preservation of design representation simplifies deployment and management of a completed system, as well as communication between developers and users of the system. It also allows a continuous approach to the system implementation, where new functionality and features are added while preserving and extending existing functionality and maintaining a model up to date.

FIG. 5 shows system infrastructure 50, which includes networked developer workstations 51A-51B (collectively referred to as developer workstations 51), user workstations 52A-52B (collectively referred to as user workstations 52), nodes 54A-54C (collectively referred to as nodes 54) and system repository 53. This invention prescribes an infrastructure that consists of networked computers, called nodes 54, each hosting an instance of this invention's run-time software 55. This run-time software 55 is a container for component instances 56A-56G (collectively referred to as component instances 56). Component instances 56 and links 57A-57D (collectively referred to as links 57) may be created and destroyed remotely, using system management tools. These tools are used to deploy complete solutions by deploying component instances 56 to specific nodes 54. Component instances 56 may be moved from node to node while preserving links 57 and configuration data. Each node 54 has up-to-date configuration data, stored locally, that has all information about component instances 56, links 57, etc. This

information allows nodes 54 to shut down and restart without any additional information required, which contributes to overall robustness of the system.

All information about system 50 is stored in the System Repository 53. System Repository 53 includes service protocol definitions, components, component instance data, links, node deployment information, etc. System Repository 53 is populated using system tools and is transparent to the user or developer. This information is not required for any of the run-time activities within the system. It can be treated as a centralized, redundant directory, and can be recreated from information stored on nodes 54.

This invention presents a new architecture for creating and managing distributed information systems, shown on FIG. 6. System development starts with the modeling phase that involves developers 61A-61B (collectively referred to as developers 61) and domain experts/users 62A-62C (collectively referred to as domain experts/users 62).

New Services are defined, by means of Service Protocols 67, using the Service Definition Tool 64. Developers 61 and domain experts 62 contribute to this phase of development. Developed service protocols are stored in the Service Protocol Repository 53A, which is part of the System Repository 53. The Service Protocol Repository is a catalog of all defined service protocols in the system. Service protocols may be exported from and imported into the Service Protocol Repository. Service protocols can be re-used from system to system.

Developers 61, in collaboration with domain experts 62, create new Components 68 that implement services based on newly defined and/or existing service protocols 67. Developers use the Component Development Tool 65 to build components 68 and to store them in Component Repository 53B. A given component 68 may implement unlimited numbers of services, both as a consumer and as a provider. Each implemented service protocol is exposed as a Service Access Port, such as service access ports 11 and 12, shown in Fig.1. The

component developer may define Configuration Attributes. Attributes are used to configure individual instances of a component 68. Component developers use attributes to alter component functionality at run-time based on the values supplied. Component Repository 53B is a catalog of all components 68 defined in the system. As with service protocols, components 68 can be shared between multiple systems.

Domain experts/users 62 utilize the System Development Tool 66 to define system behavior by creating and configuring (attributes are configured) instances of components 56A-56B (collectively referred to as component instances 56). The System Development Tool 66 stores all configuration information in the Model Repository 53C. When created, each instance 56 is given a meaningful, unique name, usually reflecting its system location and/or functionality. Component instances 56 are connected through Links 57 -- definitions of the communication channel. A link 57 can be created between two Service Access Ports if they represent two ends of the same Service Protocol 67, e.g. if the first port represents a service provider and the second port represents a complementary (inverse version of the same service protocol) service consumer. Each port may be connected to any number of complementary ports on any number of component instances 56, including the parent component instance itself.

FIG. 7 presents a deployment diagram for component instances 56 and links 57 as they are assigned to nodes 54. By creating instances 56 of the components and connecting them by links 57, the user builds a Logical System Model 71 (see FIG. 6). At this point, component instances 56 are defined but are not yet running; they have to be deployed (mapped) to the Physical System Infrastructure 72 by assigning component instances 56 to Nodes 54, using System Development Tool 66. As shown in FIG. 7, component instances 56A, 56B and 56C are deployed to node 54A, component instances 56F, 56G and 56I are deployed to node 54B, and component instances 56D and 56E are deployed to node 54C. Nodes 54 are networked together using any media that supports IP protocols.

FIG. 8 illustrates the interaction between a component instance 56 and run-time software 55. Run-time software 55 implements API (Application Programming Interface) 95, communication 94, Local System Repository 93 and Active Links Table 97. Once assigned to the node 54, component instance 56 data is downloaded to the target node and stored in the Local System Repository 93. The node's Run-time Software 55 is responsible for creating an instance 56, supplying it with configuration data and establishing links 57. Run-time software 55 is a container for component instances 56 and provides implementation of the API 95 used by the component developers 61 to access container functionality. Container functionality includes timing, scheduling, configuration data access, persistence, communication 94 and security. Run-time software 55 is dynamically changing, based on links 57 defined for the current node 54. Because component instances 56 do not have any information about peers, it is the responsibility of the run-time software 55 to manage all communication details as defined by links 57. This is in contrast with existing architectures, where run-time containers are rigid and responsibilities are shifted to the component implementation. By adding programmability features to the run-time and isolating component implementation from the system composition, this invention provides a flexible component deployment infrastructure.

The System Development Tool 66 can be used to modify configuration data for component instances 56. If a component instance 56 is deployed, these changes are sent to the node's 54 run-time environment 55, which in turn notifies the component instance 56 of the changes and provides new configuration to the instance 56. If a deployed instance 56 is deleted from the Model Repository 53C, it would be removed from the node 54, and all related data would be deleted from the Local System Repository 93. All active links 57 connected to the deleted instance 56 would be shutdown and the run-time software 55 would deny any request of connection addressed to this instance 56. Deleting

an instance 56 on one end of a link 57 automatically deletes the link 57 itself. These changes are propagated to the nodes 54 where affected component instances 56 were deployed.

New links 57 may be created at any time using the System Development Tool 66. If a link 57 is created between two deployed component instances 56, the link information is sent to the nodes 54 involved and stored in both nodes' Local System Repository 93. Run-time software 55 then creates a logical connection and starts passing messages to and from the instance's 56 port. Establishing a link 57 is a node's 54 local operation, and is not involved in any communication with the rest of the system. This ensures that system components, such as nodes 54 and system repository 53, can go on and off line without affecting overall system functionality. Note that this is only true if the off-line node 54 is not hosting any component instances 56 whose presence is required for normal system operation. Creating redundant component instances 56 and links 57 and distributing them across multiple nodes 54 can solve this problem, but this relates to the particular system design and is outside of the scope of this invention.

Although the present invention has been described with reference to preferred embodiments, workers skilled in the art will recognize that changes may be made in form and detail without departing from the spirit and scope of the invention.

CLAIM(S):

1. An architecture for developing a distributed information system, the architecture comprising:

a service definition tool for generating service protocols as a service definition, each service protocol including a plurality of messages, the messages including incoming messages and outgoing messages, each message carrying a plurality of data fields;

a component development tool for generating a first and a second plurality of components that implement and consume services, each component in the first plurality of components representing a physical entity in the distributed information system, each component in the second plurality of components representing a logical entity in the distributed information system;

a system development tool for generating a plurality of component instances based on the first and the second plurality of components; and

an engine software program for running on each of a plurality of networked nodes, the engine software program providing a programmable run-time environment for hosting the plurality of component instances and supporting communication between component instances.

2. The architecture of claim 1, wherein the services are defined by the service protocols, and wherein the component development tool allows implementations of the services in the first and the second plurality of components, the implementations of the services exposed as service provider ports and service consumer ports.

3. The architecture of claim 2, wherein service provider ports and service consumer ports based on the same service protocol are complimentary.

4. The architecture of claim 3, wherein the system development tool allows communication links to be defined between service provider ports and complimentary service consumer ports.

5. The architecture of claim 1, wherein at least one component instance is self-sufficient and functions without interacting with other component instances.

6. The architecture of claim 4, wherein a component instance includes at least one service provider port that allows multiple simultaneous communication links with complimentary service consumer ports.

7. The architecture of claim 4, wherein component instances are executed concurrently, and wherein the communications between service provider ports and complimentary service consumer ports are asynchronous.

8. The architecture of claim 4, wherein the communication links include one-to-one links, one-to-many links and many-to-many links, regardless of ports involved.

9. The architecture of claim 1, wherein the system development tool allows each component instance to be configured.

10. The architecture of claim 1, wherein the system development tool represents the distributed information system as a single entity, regardless of physical node and network composition into which the component instances will be deployed.

11. The architecture of claim 10, wherein the system development tool deploys each component instance to one of the plurality of networked nodes.

12. The architecture of claim 11, and further comprising a local repository on each of the plurality of nodes, the local repository on each node storing data representing the component instances deployed to and hosted by that node and storing communication link data for the component instances deployed to and hosted by that node.

13. The architecture of claim 11, wherein the system development tool allows changes to be made to the component instances deployed to and hosted by the plurality of networked nodes and allows changes to be made to communication links between the component instances deployed to and hosted by the plurality of networked nodes.

14. The architecture of claim 13, wherein the system development tool allows deletion of the component instances deployed to and hosted by the plurality of networked nodes and allows deletion of communication links between the component instances deployed to and hosted by the plurality of networked nodes.

15. The architecture of claim 1, and further comprising a central system repository for storing the components, the component instances, link data, infrastructure configuration and configuration data for the service protocols.

16. The architecture of claim 1, wherein at least one of the component instances supports continuous activities internally.

17. The architecture of claim 1, wherein each of the component instances is configurable to participate in activities that are collectively performed by a plurality of component instances.

18. The architecture of claim 4, wherein the only run-time dependencies between component instances that communicate with each other are logical dependencies implemented using the component development tool.

19. A distributed information system comprising:

a plurality of component instances connected by links; and

a plurality of networked nodes running an engine software program, the engine software program providing a programmable run-time environment for hosting the plurality of component instances and supporting communication between component instances.

20. The distributed information system of claim 19, wherein the run-time environment acts as a container for the component instances, and wherein the run-time environment provides API implementations to the component instances.

21. The distributed information system of claim 19, and further comprising a local repository on each of the plurality of nodes, the local repository on each node storing component instances deployed to and hosted by that node and storing communication link data for component instances deployed to and hosted by that node.

22. The distributed information system of claim 21, wherein the local repository notifies component instances and links of configuration data changes.

23. The distributed information system of claim 20, wherein the run-time environment dynamically manages ports and links for the component instances and includes communication implementation for delivering messages between connected complimentary component instance ports.

24. The distributed information system of claim 19, wherein the engine software program supports communication with a system development tool that can be used to create, destroy and modify component instances and links.

25. The distributed information system of claim 21, wherein the programmable run-time environment dynamically changes according to configuration data stored in the local repository.

26. A method of modeling a distributed information system, the method comprising:

identifying physical and logical entities in the distributed information system;

organizing the identified entities into a hierarchical system model;

identifying services provided by the identified entities using the hierarchical system model;

creating service protocols, each service protocol defining one of the identified services;

implementing the identified services as a plurality of components using the service protocols, the plurality of components implemented as concurrent independent software entities, the implementation including generating service provider ports and service consumer ports to expose the implemented services;

populating the system model by creating instances of the implemented components;

linking the component instances together; and

assigning the component instances to a plurality of run-time software engines.

27. The method of claim 26, wherein the organization of the entities in the hierarchical system model is based on the generality and interdependence of the entities using a rule that assigns more generic entities lower in the hierarchical model than more specific entities.

SYSTEM AND METHOD FOR GENERATING DISTRIBUTED INFORMATION SYSTEMS

ABSTRACT OF THE DISCLOSURE

An architecture for developing a distributed information system comprises a service definition tool for generating service protocols as a service definition. Each service protocol includes a plurality of messages. The messages include incoming messages and outgoing messages. Each message carries a plurality of data fields. A component development tool generates a first and a second plurality of components that implement and consume services. Each component in the first plurality of components represents a physical entity in the distributed information system. Each component in the second plurality of components represents a logical entity in the distributed information system. A system development tool generates a plurality of component instances based on the first and the second plurality of components. An engine software program runs on each of a plurality of networked nodes. The engine software program provides a programmable run-time environment for hosting the plurality of component instances and supporting communication between component instances.

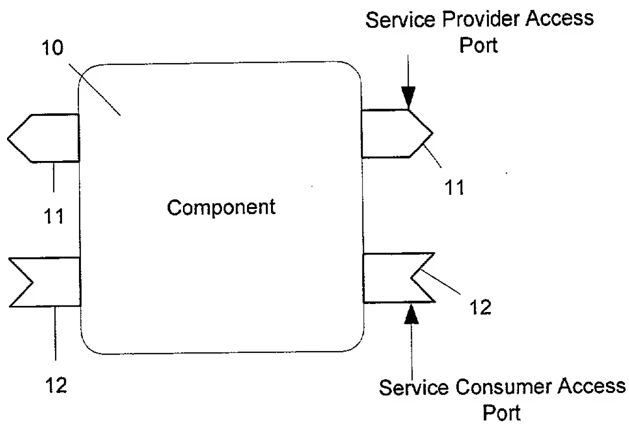


Fig. 1

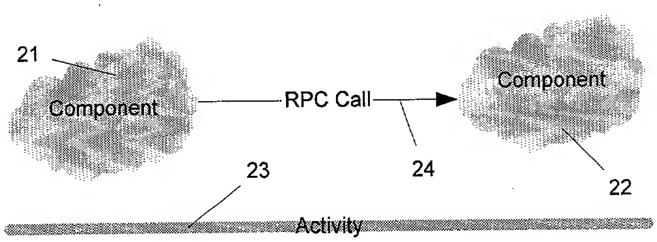


Fig. 2 (Prior Art)

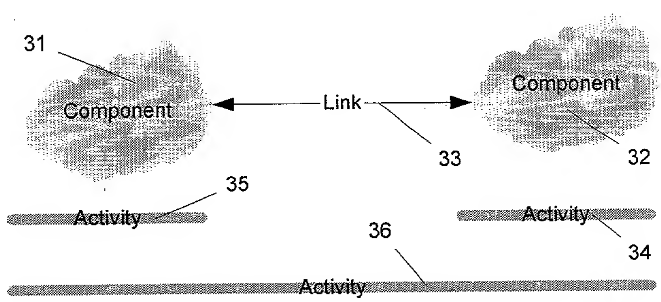
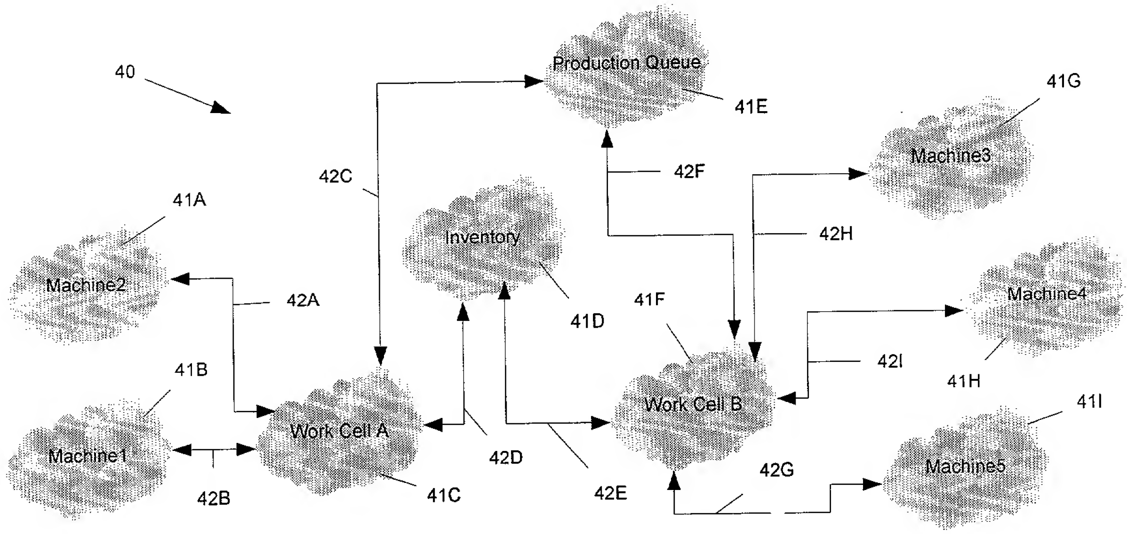
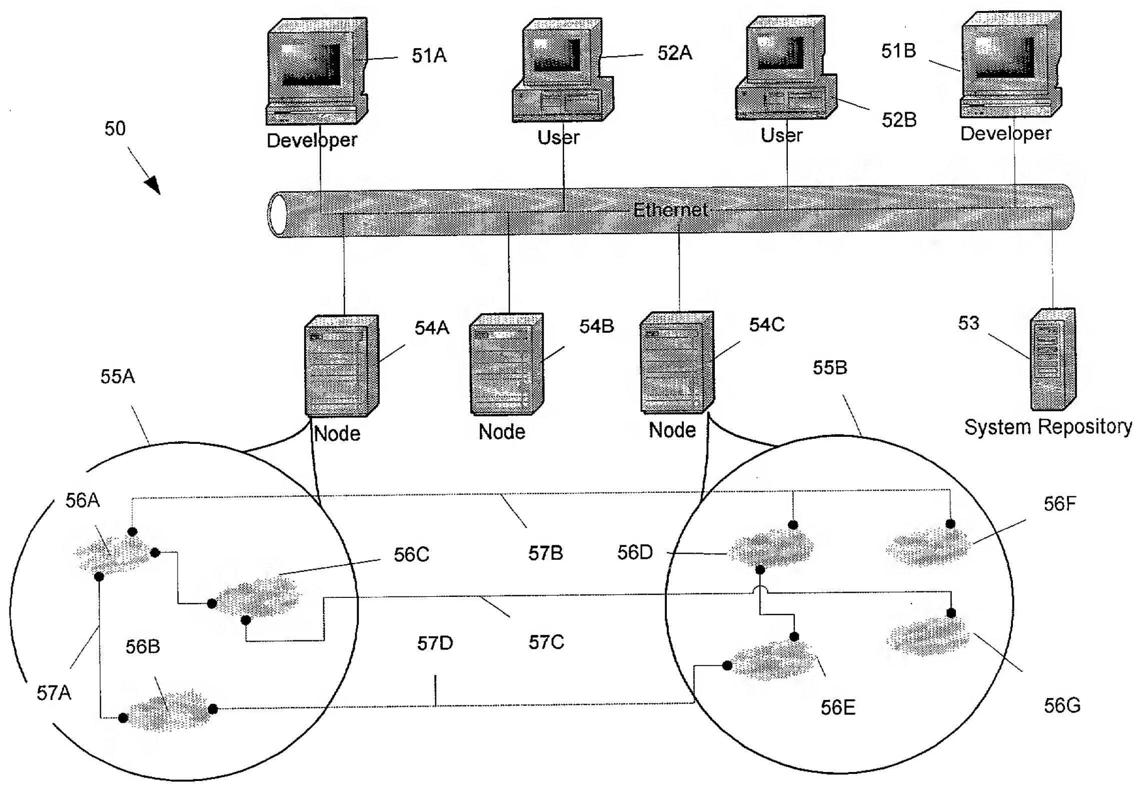


Fig. 3



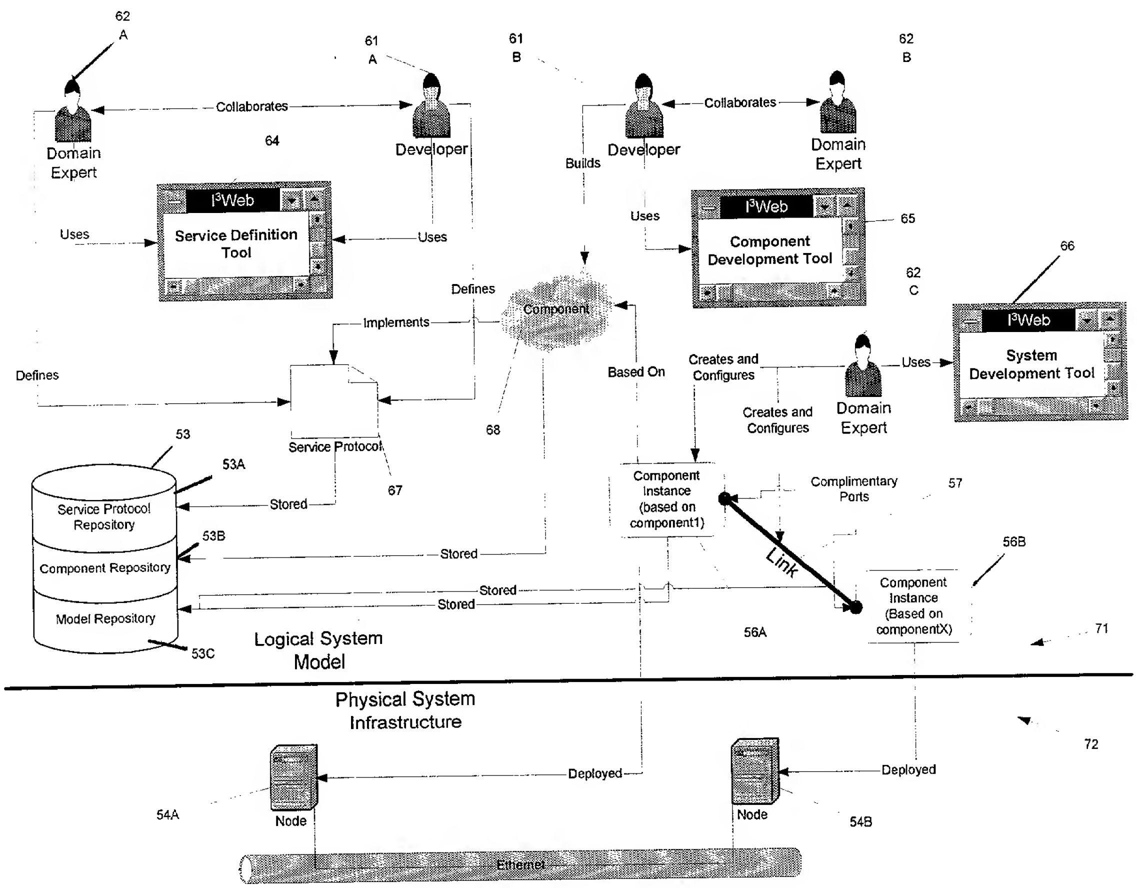
40
Production Queue
41E
41G
Machine3
42C
42F
41A
Inventory
42H
Machine2
42A
41F
Machine4
41D
42I
41B
41H
Work Cell B
41I
Work Cell A
Machine1
42D
42G
Machine5
42B
41C
42E

Fig. 4



51A
52A
51B
52B
Developer
User
User
Developer
50
Ethernet
54A
54B
54C
53
55A
55B
Node
Node
Node
System Repository
56A
56F
56C
57B
56D
56B
57D
57C
57A
56E
56G

Fig. 5



62 A
61 A
61 B
62 B
Collaborates
Collaborates
Domain Expert
Developer
Developer
Domain Expert
64
Builds
I³Web
Service Definition Tool
Uses
Uses
Uses
I³Web
Component Development Tool
65
66
62 C
Defines
Component
Implements
I³Web
System Development Tool
Uses
Defines
Based On
Creates and Configures
Creates and Configures
Domain Expert
68
53
Service Protocol
53A
67
Component Instance (based on component1)
Complimentary Ports
57
Service Protocol Repository
Stored
53B
Component Repository
Stored
Link
56B
Component Instance (Based on componentX)
Stored
Stored
Model Repository
53C
Logical System Model
56A
71
Physical System Infrastructure
72
Deployed
Deployed
54A
Node
Node
54B
Ethernet

Fig. 6

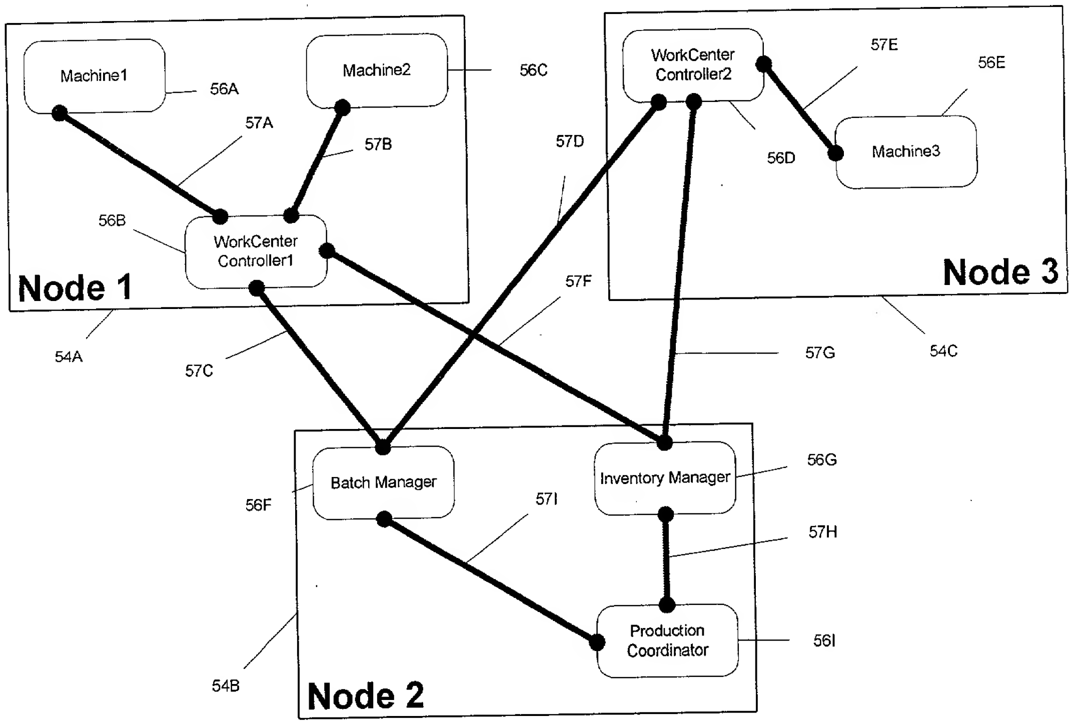

Machine1
56A
Machine2
56C
57A
57B
56B
WorkCenter Controller1
Node 1
54A
57C
WorkCenter Controller2
57E
56E
56D
Machine3
57D
Node 3
57F
57G
54C
Batch Manager
Inventory Manager
56G
56F
57I
57H
Production Coordinator
56I
54B
Node 2

Fig. 7

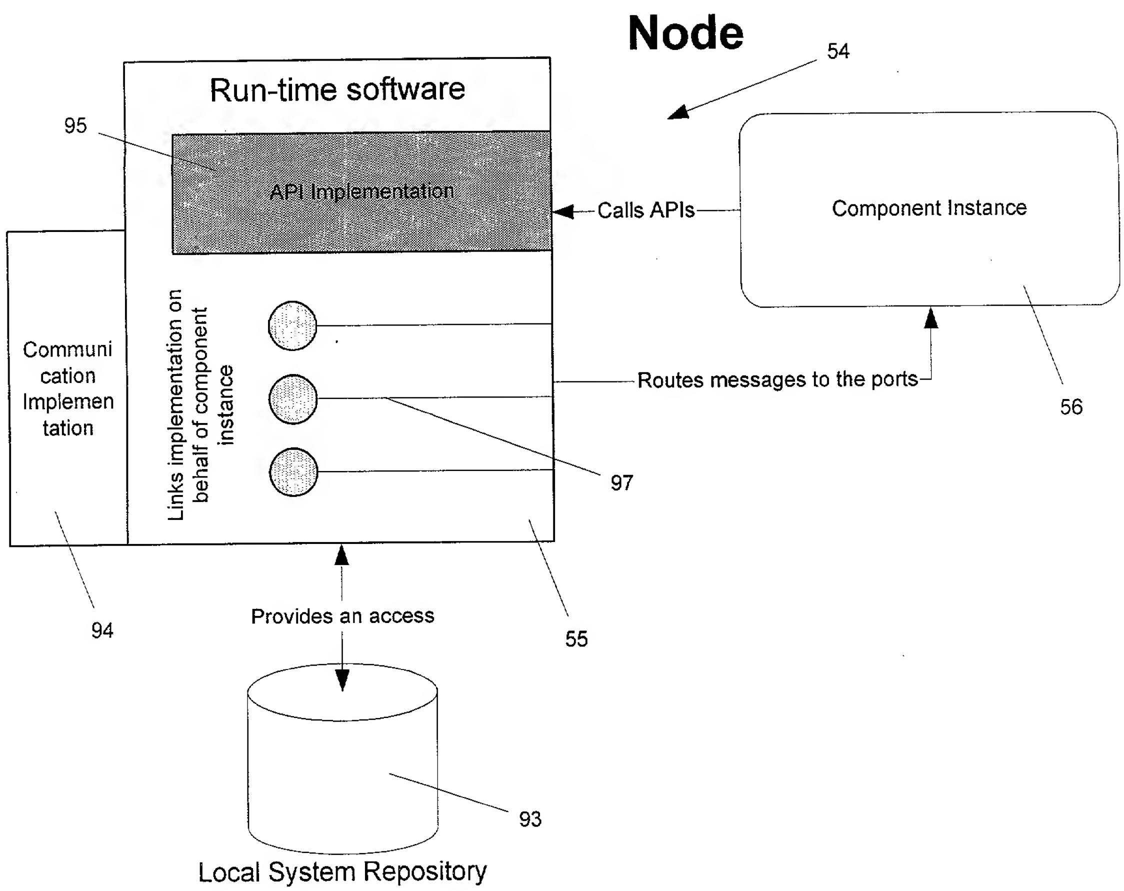

Node
54
Run-time software
95
API Implementation
Calls APIs
Component Instance
Communi cation Implemen tation
Links implementation on behalf of component instance
Routes messages to the ports
56
97
94
Provides an access
55
93
Local System Repository

Fig. 8

<table>
<tr><td colspan="2" rowspan="3">DECLARATION FOR UTILITY PATENT APPLICATION (37 C.F.R. 1.63)</td><td>Attorney Docket No.</td><td>N298.12-0001</td></tr>
<tr><td>First Named Inventor</td><td>Michael Feldman</td></tr>
<tr><td colspan="2">COMPLETE IF KNOWN</td></tr>
<tr><td rowspan="4">X Declaration Submitted with Initial Filing</td><td rowspan="4">_ Declaration Submitted after Initial Filing (Surcharge (37 C.F.R. 1.16(e)) Required)</td><td>Application Number</td><td></td></tr>
<tr><td>Filing Date</td><td>August 15, 2000</td></tr>
<tr><td>Group Art Unit</td><td></td></tr>
<tr><td>Examiner Name</td><td></td></tr>
</table>

As a below named inventor, I hereby declare that my residence, post office address, and citizenship are as stated below.

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:
SYSTEM AND METHOD FOR GENERATING DISTRIBUTED INFORMATION SYSTEMS

the specification of which:
x is attached hereto OR
_ was filed on ________ as United States Application Number ___________or PCT International Application Number ____
and was amended on ___________ (if applicable).

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment specifically referred to above.

I acknowledge the duty to disclose information which is material to patentability as defined in 37 C.F.R. 1.56.

I hereby claim foreign priority benefits under 35 U.S.C. 119(a)-(d) or 365(b) of any foreign application(s) for patent or inventor's certificate, or 365(a) of any PCT international application which designated at least one country other than the United States of America, listed below and have also identified below, by checking the box, any foreign application for patent or inventor's certificate, or of any PCT international application having a filing date before that of the application on which priority is claimed.

Prior Foreign Application Number(s)	Country	Foreign Filing Date (MM/DD/YYYY)	Certified Copy Attached? Yes No

I hereby claim the benefit under 35 U.S.C. 119(e) of any United States provisional application(s) listed below.

Application Number(s)	Filing Date (MM/DD/YYYY)
60/149,507	08/17/1999

I hereby claim the benefit under 35 U.S.C. 120 of any United States application(s), or 365(c) of any PCT international application designating the United States of America, listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States or PCT international application in the manner provided by the first paragraph of 35 U.S.C. 112, I acknowledge the duty to disclose information which is material to patentability as defined in 37 C.F.R. 1.56 which became available between the filing date of the prior application and the national or PCT international filing date of this application.

U.S. Parent Application or PCT Parent Number	Parent Filing Date (MM/DD/YYYY)	Parent Patent Number (*if applicable*)

DIRECT ALL CORRESPONDENCE TO:

Name	Jeff A. Holmen
Address	Kinney & Lange, P.A. THE KINNEY & LANGE BUILDING 312 South Third Street Minneapolis, Minnesota 55415 United States
Telephone	(612) 339-1863
Fax	(612) 339-6580

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. 1001 and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Sole or First Inventor:

Given Name (First and middle (if any))	Family Name or Surname
Michael	Feldman

Inventor's signature		Date:
Residence	Minnetonka, Minnesota	Citizenship: USA
Post Office Address	5602 Pompano Drive	
City, State, Country	Minnetonka, Minnesota 55343, USA	